.NET Full Stack Development Program

—

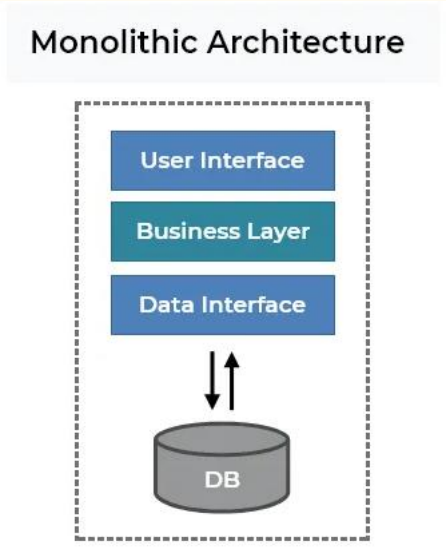
Day 30 Introduction to Microservice

Outline

- Application Architecture
  - Monolithic
  - Microservices
- Microservices
  - Service Discovery
  - Service Communication
  - API Gateway
- Microservices Deployment

Monolithic

- We put everything in one application



Monolithic

- This architecture has a number of benefits:
  - Simple to develop - the goal of current development tools and IDEs is to support the development of monolithic applications
  - Simple to deploy - we simply need to deploy the monolithic application as a whole
  - Simple to scale - we can scale the application by running multiple copies of the application behind a load balancer

Monolithic

- Ways to improve performance
  - Multi-Threading
  - Offline Processing
- But what if the number of requests sent to our application increases and our application reaches its limit?

Monolithic

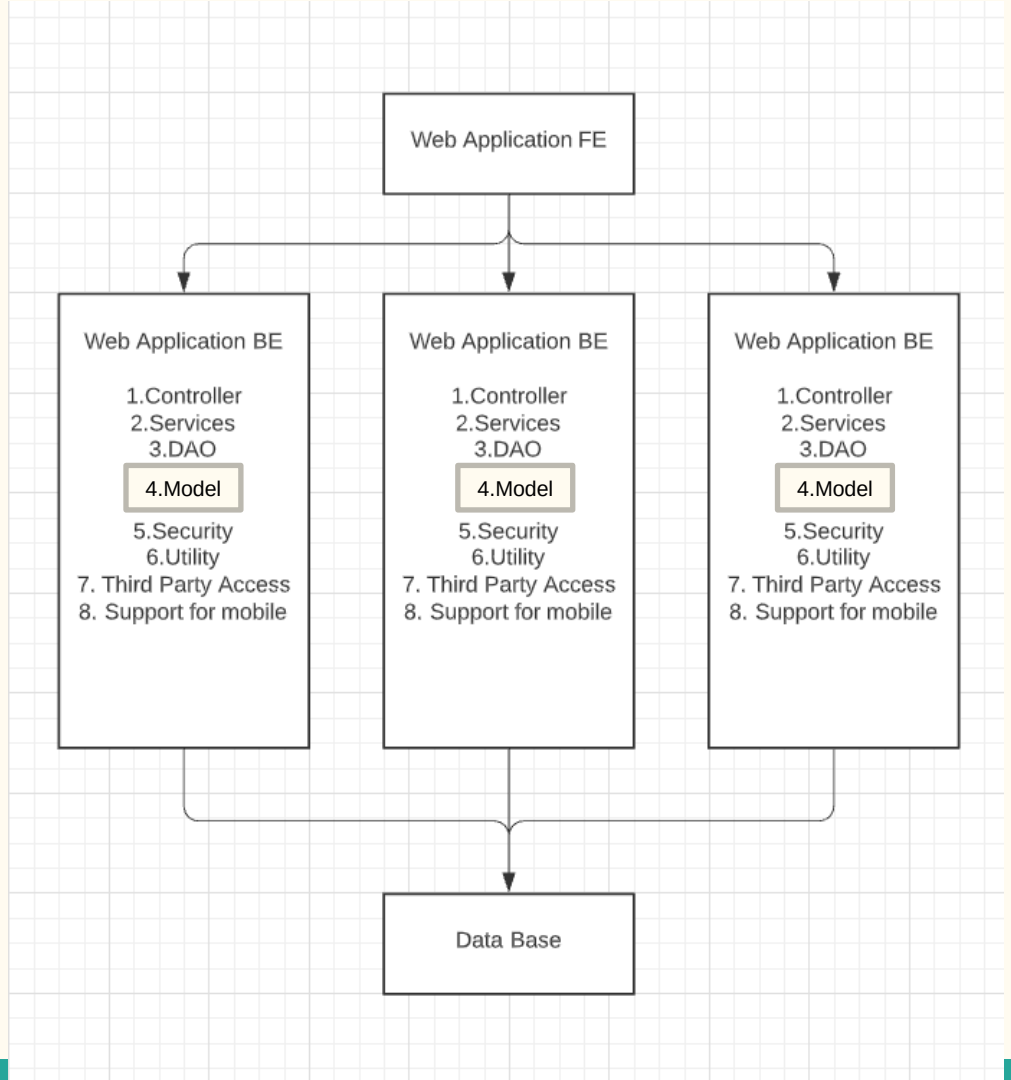
- Scaling
    - Vertical: adding more power to an existing machine. For example, more powerful CPU or more RAM.
    - Horizontal: deploying multiple instances of our application.

Monolithic

- Vertical Scaling (not preferred)
  - More powerful machine is more expensive.
  - Always have a limit
    - What if the most powerful machine is still not powerful enough?

Monolithic

- Horizontal Scaling
    - Though cost money, but cheaper than vertical scaling. Multiple normal performance machines are cheaper than one high performance machine.
    - We can have as many instances as we want. The limit is the resources needed to make those machines.


Web Application FE
Web Application BE
1.Controller
2.Services
3.DAO
4.Model
5.Security
6.Utility
7. Third Party Access
8. Support for mobile
Web Application BE
1.Controller
2.Services
3.DAO
4.Model
5.Security
6.Utility
7. Third Party Access
8. Support for mobile
Web Application BE
1.Controller
2.Services
3.DAO
4.Model
5.Security
6.Utility
7. Third Party Access
8. Support for mobile
Data Base

Horizontal Scaling

How does the client-side application know which server-side instance to request?

Load Balancer Scenario

- Imagine your web application is a table in a restaurant, and each table is an instance of the application that can serve a customer(**handle a request**).

- The load balancer in this example will be the host that waits at the restaurant's entrance and leads the customers(**request**) to an empty table(**idling application instance**).

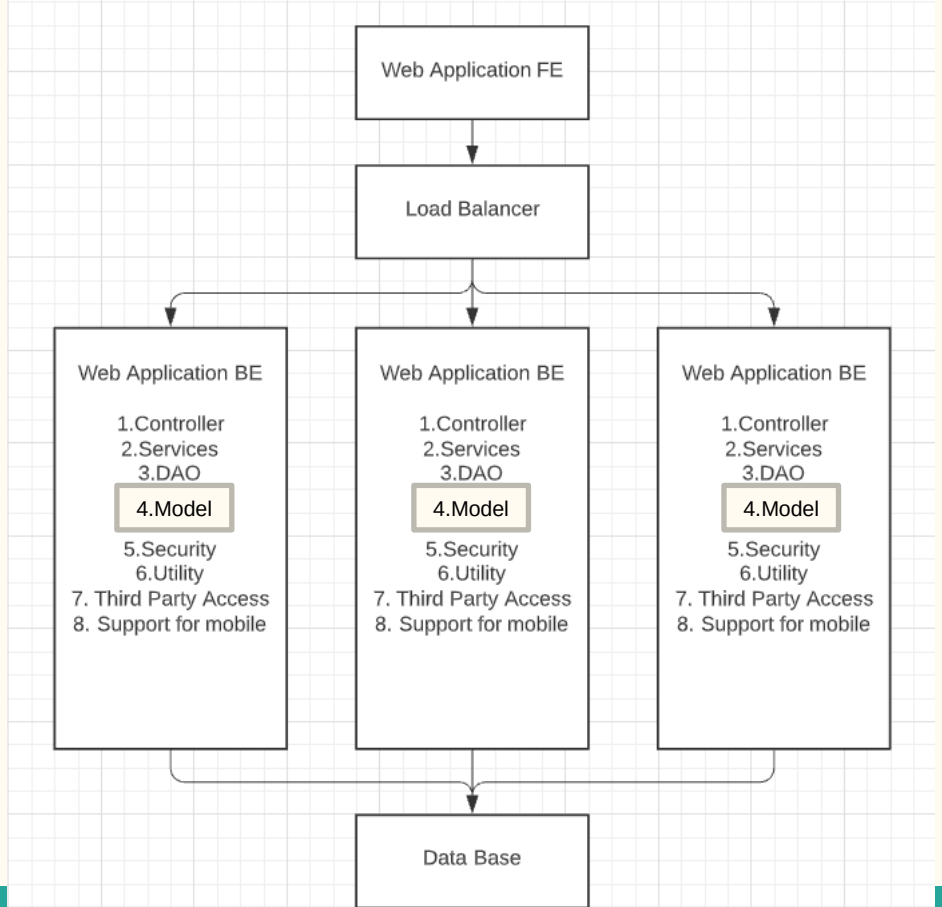
- The host(**loader balancer**) can bring up more tables(**instances of the application**) if the number of customers(**requests**) hits the current limit.

Monolithic

- Load Balancer
  - Load balancing – efficiently distributing incoming network traffic across a group of backend servers.
  - Load balancer acts as a traffic distributer sitting in front of all your backend servers.
  - It will route your client request to the most appropriate backend server which maximize speed, capacity utilization and ensures no server is over-worked. No impact on the performance.

Load Balancer

- Can be both software and hardware.
- Can be performed on both the client side and server side.
- Type: Application, Network, Classic
- Algorithm: Round-Robin, Weighted Round-Robin...
- Technologies: Nginx, HAProxy, Microsoft Azure Load Balancer.


Web Application FE
Load Balancer
Web Application BE
1.Controller
2.Services
3.DAO
4.Model
5.Security
6.Utility
7. Third Party Access
8. Support for mobile
Web Application BE
1.Controller
2.Services
3.DAO
4.Model
5.Security
6.Utility
7. Third Party Access
8. Support for mobile
Web Application BE
1.Controller
2.Services
3.DAO
4.Model
5.Security
6.Utility
7. Third Party Access
8. Support for mobile
Data Base

Monolithic

- However, once the application becomes too large, and the team grows in size, there are problems:
  - The large monolithic code base intimidates developers, especially ones who are new to the team
  - Overloaded IDE and web container
  - Might be scaling up the entire application just for one service.
  - Take a long time to deploy.
  - Requires a long-term commitment to a technology stack (Hard to adopt new technologies)
  - One error takes down the entire application.

Microservice

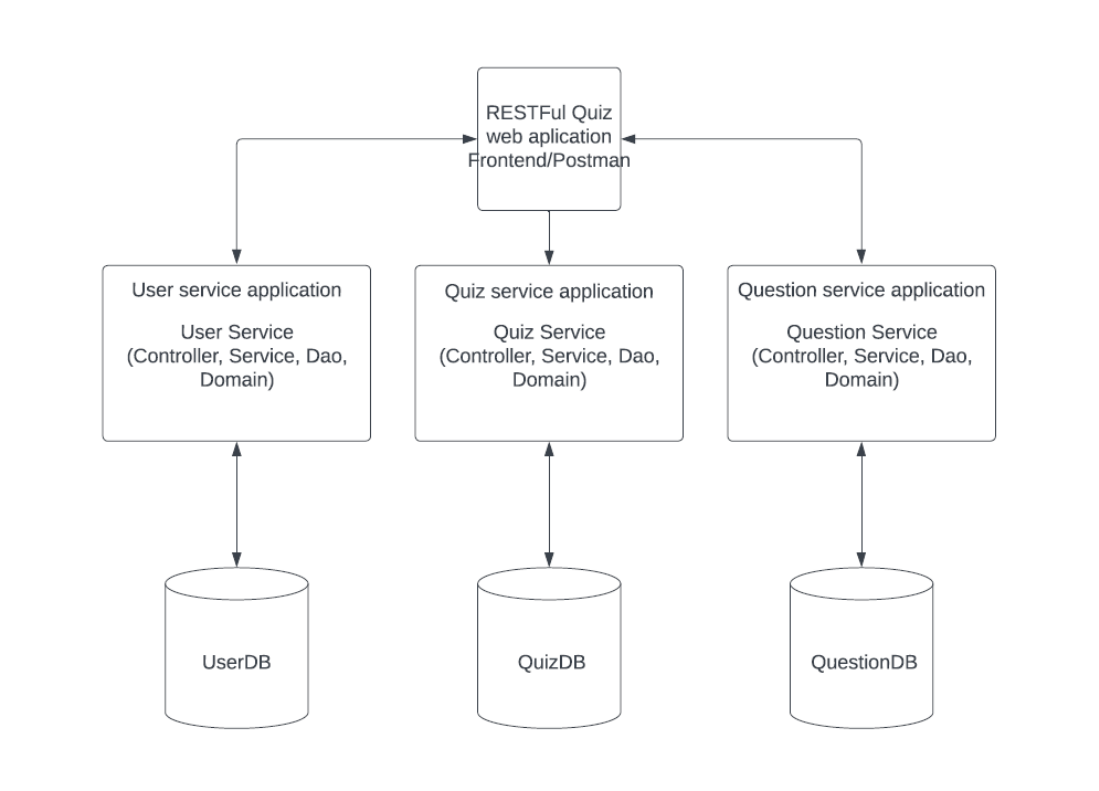
- What are Microservices?
    - Microservices, also known as microservice architecture, it's an architectural style that structures an application as a collection of small services.
- Microservices are loosely coupled, services connect to each other on a service endpoint, and they don't need to know the internal implementations of the other.
- Every Microservices are managed by independent teams that give the autonomy to decide when to release a feature or fix a bug without waiting for the entire team's release cadence.

Microservices

- Here are some key characteristics of microservices:
  - Microservices are small, independent, and **loosely coupled**.
  - Each microservice has a **separate codebase**, which can be managed by a small development team.
  - Microservices are **deployed independently**. A team can update an existing microservice without rebuilding and redeploying the entire application.
  - Microservices are responsible for persisting their data or external state in their respective databases. Unlike the monolithic architecture, **microservices don't share databases**.
  - Microservices **communicate** with each other by **using well-defined APIs**. Internal implementation details of each service are hidden from other services.
  - Supports polyglot programming. For example, microservices **don't need to share the same technology stack**, **libraries**, **or frameworks**.

Microservices

- The cons of microservices
  - The implementation of Microservice is complex
  - The services are loosely coupled, so the data could be duplicated
  - API calls need to be resilient
  - Maintenance could be complicated

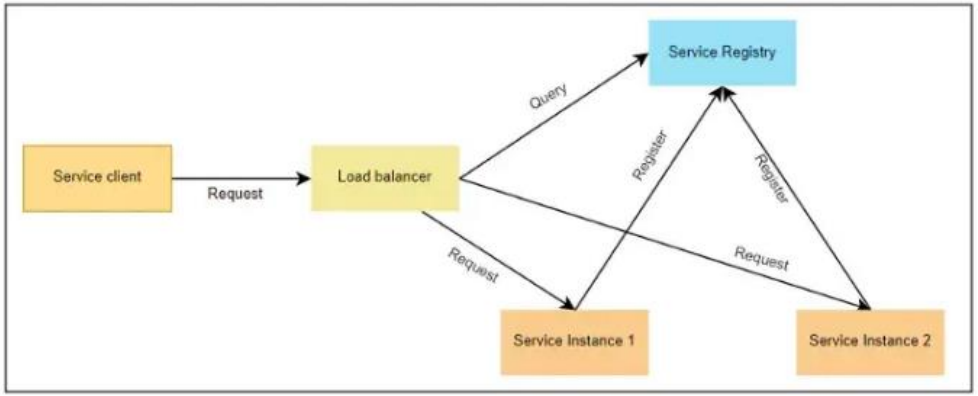

RESTFul Quiz
web aplication
Frontend/Postman
User service application
User Service
(Controller, Service, Dao,
Domain)
Quiz service application
Quiz Service
(Controller, Service, Dao,
Domain)
Question service application
Question Service
(Controller, Service, Dao,
Domain)
UserDB
QuizDB
QuestionDB

Microservices

- What if a service needs to communicate with another service?
- We keep the locations (hostname and port number) of all services in every service?
- We keep the locations of all services in a central place?

Microservices

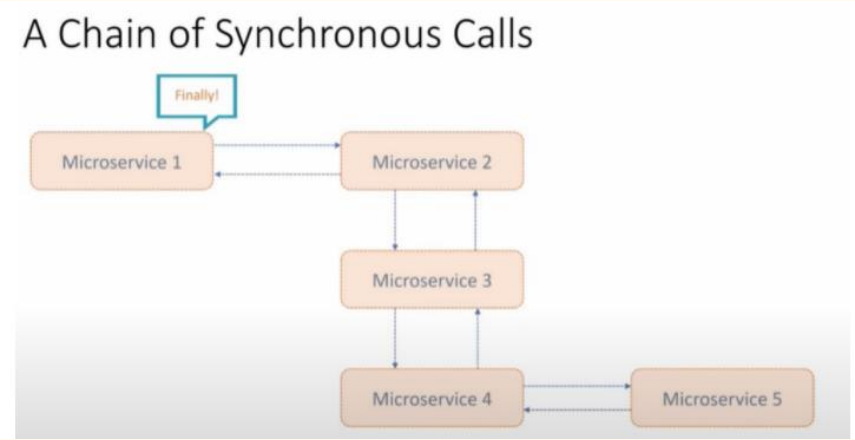
- Service Discovery/Registry: Keep track of services and service addresses and endpoints.
  - Steeltoe
  - Consul
  - Docker

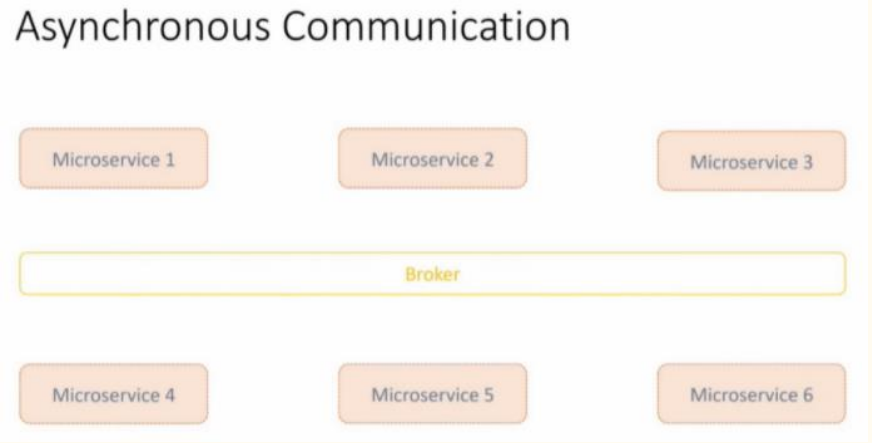


Service Communication

- Synchronous communication
  - REST
  - gRPC
  - SignalR
- Asynchronous communication
  - Azure Service Bus, Messaging(Rabbit MQ, Kafka)
    - Often preferred way for inter-service communication

Synchronous vs Asynchronous Communication





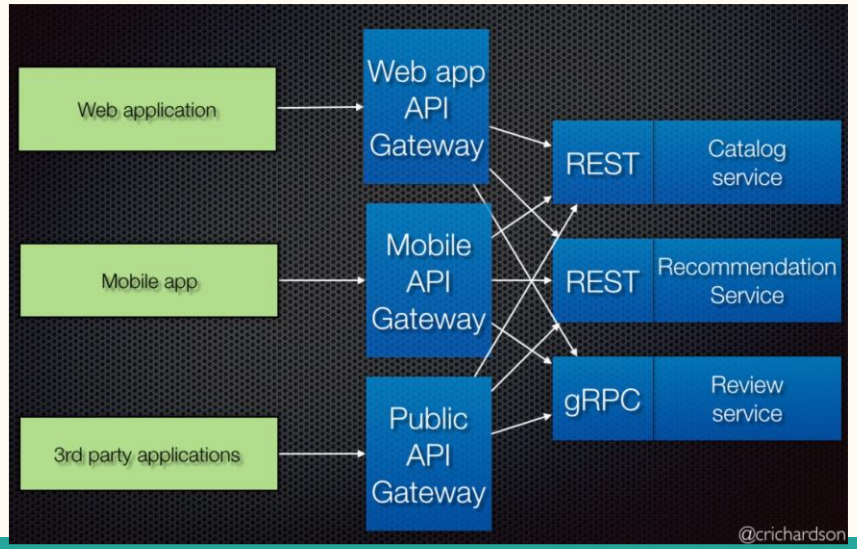
API Gateway

- What if we have 10 services, and each service provides 10 entry points, so we will have 100 different URLs?

- How will the front end know which endpoint to call for a specific service?

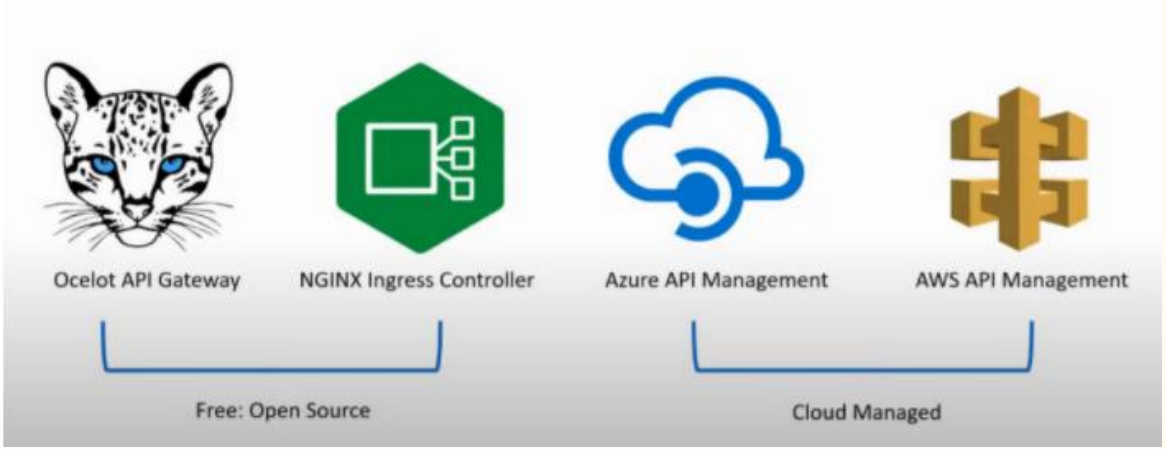
API Gateway

- API gateway is the single-entry point for all clients
- Work as a proxy service to route a request to the backend microservices
- Aggregate the results to send back to the consumer



API Gateway

- When it comes to choosing an API gateway for our services, there are a variety of options:



Different type of services

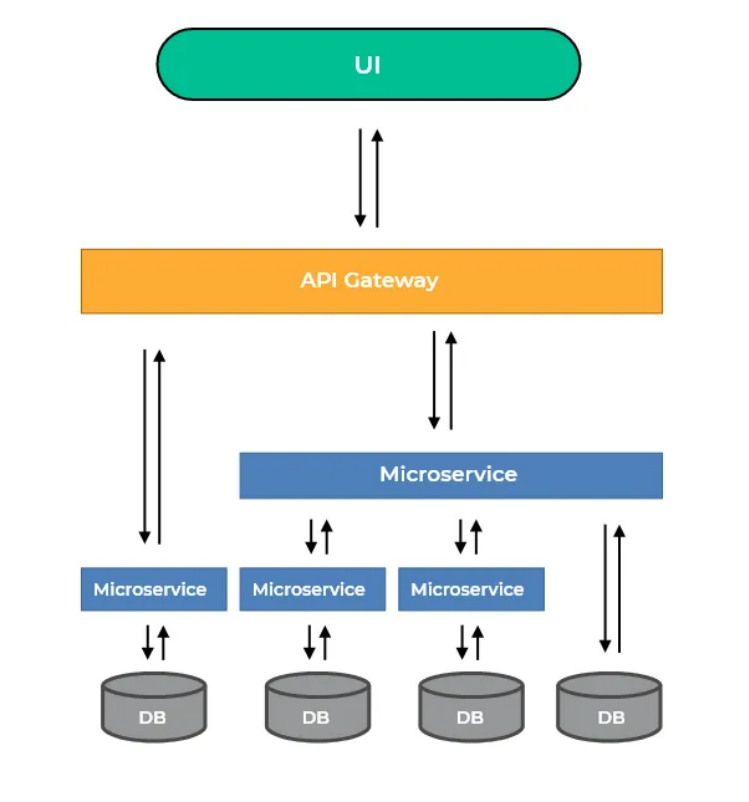
Core Services:

- Core services are the fundamental building blocks of a microservice-based application. These services provide the basic infrastructure for building and deploying microservices and are usually low-level.

Composite Services:

- Composite services, on the other hand, are built on top of core services and provide additional functionality and capabilities to the microservice architecture. These services are designed to address specific business needs and provide higher-level functionality.

Microservice Architecture



Microservices

- How are Microservices Packaged and Deployed?
  - Docker
  - Kubernetes

Microservices

- Containerization -- Docker
  - Containerization is an approach to software development in which an application or service, its dependencies, and its configuration (abstracted as deployment manifest files) are packaged together as a container image. You can test the containerized application as a unit and deploy them as a container image instance to the host operating system (OS).
  - Just as the shipping containers allow goods to be transported by ship, train, or truck. Software containers act as a standard unit of software deployment that can contain different code and its dependencies.

Docker

- What is docker?
    - Docker is an open-source project for automating the deployment of applications as portable, self-sufficient containers that can run in the cloud or on-premises. Docker is also a company that promotes and evolves this technology, working in collaboration with cloud, Linux, and Windows vendors, including Microsoft.

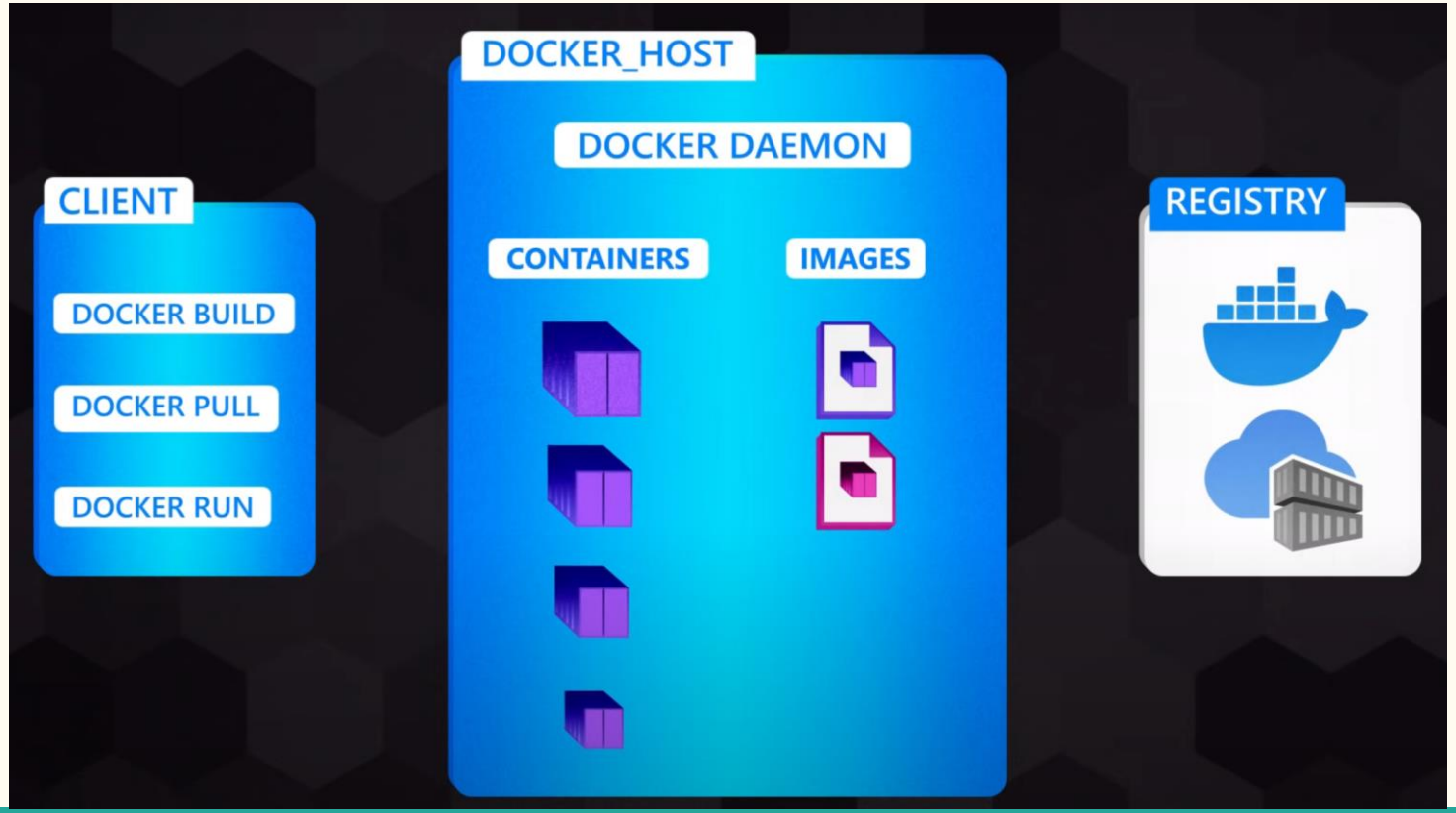
    - Docker containers can run anywhere on Azure: on-premises in the customer's datacenter, in an external service provider, or in the cloud. Docker image containers can run natively on Linux and Windows.

Docker Image

- What is an image
  - When a developer uses Docker, they create an app or service and package it and its dependencies into a container image. An image is a static representation of the app or service and its configuration and dependencies.

  - It's this image that, when run, becomes our container. The container is the in-memory instance of an image.

  - A container image is immutable. Once you've built an image, the image can't be changed. Since you can't change an image, if you need to make changes, you'll create a new image. This feature is our guarantee that the image we use in production is the same image used in development and QA.


DOCKER_HOST
DOCKER DAEMON
CONTAINERS
IMAGES
CLIENT
DOCKER BUILD
DOCKER PULL
DOCKER RUN
REGISTRY

Microservices Deployment

- Here are some general steps for deploying microservices with Docker:

1. Create a .NET Core application: Start by creating a .NET Core application that you want to deploy as a microservice. You can use Visual Studio or the .NET CLI to create a new .NET Core project.

2. Create a Dockerfile: A Dockerfile is a script that contains instructions for building a Docker image. Create a Dockerfile that includes instructions for building an image of your .NET Core microservice.

3. Build a Docker image: Use the Docker CLI to build an image of your .NET Core microservice based on the Dockerfile you created.

4. Push the Docker image to a registry: Push the Docker image you built to a Docker registry, such as Docker Hub or a private registry.

5. Deploy the Docker image: Deploy the Docker image to a Docker host, such as a virtual machine or a cluster of machines running in the cloud. You can use tools such as Docker Compose or Kubernetes to deploy the Docker image and manage the containers that run your .NET Core microservice.